Security Lab – Hacking-Lab Challenges Part 1

Introduction

In this lab, you'll do your first three Hacking-Lab challenges. The goal is to learn about working on the challenges in general and to learn about some vulnerabilities and corresponding attacks. To access the challenges, select event *Challenges Part 1* in the Hacking-Lab – this shows you the three challenges that are part of this event. It's recommended to work through the challenges in the given order.

Note that all three challenges are in the context of web applications. In future lectures and security labs in module SWS1, you'll learn much more about vulnerabilities in web applications and corresponding attacks. However, you should still be able to do the three challenges in this security lab with the information and hints that are provided here. Note also that the fundamental vulnerabilities and attacks discussed here are not limited to web applications, but may also exist in the context of other systems and application types.

General Information about how to work on the Challenges

The following describes some general information about how to work on the challenges. It's not only valid for this security lab, but also for all future Hacking-Lab based security labs that you'll do in module SWS1.

To do a specific challenge, first select the challenge. The challenge contains a description, which you should always read. To start the resource (i.e., the vulnerable server) for a specific challenge, use the *Resources* box at the top right. With the arrow, the resource is started, which opens a *Resource Properties* popup that shows you the hostname/URL of the resource. To access the resource, click the icon next to *Open* or copy the hostname/URL into the address bar of the Firefox browser in your Hacking-Lab VM.

If the hostname looks random as in, e.g., *5d0fc42b-39e5-46f7-ac9b-697d7936e4c9.idocker.vuln.land*, then this is a personal resource only for you. It runs for one hour after which it will be automatically terminated. If you still need access after one hour, start the resource again. You can also stop (and restart) a resource manually at any time. Be aware that whenever you restart a resource, it has a different hostname and all changes that were made are lost.

If, on the other hand, the resource does not have a random part as in, e.g., *pwspray.vm.vuln.land*, then the resource is shared among all Hacking-Lab users (and it is always running). In general, when using a shared resource, activities of one user may have an effect on other users. For instance, if many users do aggressive attacks (e.g., by using a tool) against a such a resource at the same time, there may be problems with reachability or performance of the resource. This implies that you should use aggressive tools with care, and only if really required. In general, if you experience significant problems when working on a challenge that uses a shared resource, it's best to move on to the next challenge and come back to the previous one when it is «less crowded».

Once you have read the challenge description and started the resource, read the description in the SWS1 security lab assignment, i.e., in the lab descriptions (as this one here) found on Moodle. Be aware that these descriptions may be somewhat different than the challenge description. If there are differences, the description in the SWS1 security lab assignment always has precedence. The description in this document always describes the *goal* you should achieve, the *required solution for full points* (each challenge can give you 50, 100, 200 or 300 Hacking-Lab points), and (in most cases) some *hints*. You may try to solve a challenge without these hints, but this may be hard.

To submit your solution (according to the *required solution for full points*), enter it using *SUBMIT SOLUTION* near the top of the challenge description. Try to keep your solutions short and to the point, but always include all relevant information (i.e., how you detected the vulnerability and what attack steps and specific attack data (e.g., injected strings) you used. In general, your solution should allow to reproduce the attack. The instructor will review your solution and grant the points, if your answer is sound. Otherwise, you'll get some feedback (and maybe parts of the points), which you can inspect under *SOLUTION HISTORY*. In this case, you can provide additional information, which may grant you full points. In general, please make sure to only submit a solution once you are convinced that you get full points. Note also that only one lab team member has to submit the solution as the lab points are awarded per team - in fact, having only one team member submitting the solution will make the task for your instructor when checking the solutions easier, thank you!

Note that some of the challenges that you'll do are directly based on what you have learned in the lecture. However, some challenges also require attack strategies that go beyond what was discussed in the lecture. In the latter case, you'll typically get more hints that should still allow you to successfully solve such challenges.

1 Simple Encoder

1.1 Goal

Get access to a secret flag (which is basically a random-looking string in a file on the server) by exploiting a combination of two vulnerabilities: One of them exposes sensitive internal information to the browser and the other allows a path traversal attack.

This challenge shows you several things. First of all, make sure to not expose sensitive internal information, as this may be abused by attackers. Second, prevent basic vulnerabilities such as path traversal, as this may be abused to get access to arbitrary files. And third, it's sometimes needed to exploit a combination of vulnerabilities to successfully carry out an attack, as in the case here.

1.2 Required solution for full points

- A description of the attack you used to get access to the secret flag.
  As a reminder: your attack description should always (also with further challenges you'll do) allow the instructor to understand how you detected the vulnerability and what specific attack steps (including the attack strings) you used.
- The secret flag.

1.3 Hints

- Whenever starting with a challenge, play around with the application so you get a feeling and good understanding about its functionality and behavior.
- Also, it's always a good idea to analyze requests, responses, cookies etc. for hints about possible attack vectors. One way to this is by using the Firefox *Web Developer Tools*.
- In this case, check out the cookie, which has a very non-typical name. Do you recognize the encoding scheme used for the cookie value? If you see an encoded cookie, always try to decode it to understand what's behind the encoded value. Doing this should give you an idea about what the value is likely representing. Note: There may be trailing *%3D* in the cookie value. These correspond to the "=" character, URL-encoded. Simply replace *%3D* with = in this case before decoding the entire cookie value.
- To encode and decode strings, there are many online tools available. A very nice one is *CyberChef*[1]. Just enter the string to be encoded or decoded in the *Input* field and add the desired encoding scheme from the *Operations* to the *Recipe* field.
- It's also a good idea to observe the behavior of the application if the cookie is used or not used. Therefore, compare the results you get when submitting a request with or without a cookie. You should be able to see the difference by looking at the webpage as it is shown in the browser. Based on this observation, you should be able to make a connection between the value of the cookie and the information included in the (visible) webpage.

[1] *https://gchq.github.io/CyberChef*

- Now you should be able to access the secret flag. To find the name of the file that contains the flag, consider the two hints in the challenge description. As a final hint: Using *../* in a file path identifies the parent directory.

2 SCHOGGI: API Mass Assignment

2.1 Goal

Exploit a so-called mass assignment vulnerability to change the account of a standard user to an *admin* account.

To illustrate this type of vulnerability, assume that product managers in an e-shop web application are allowed to modify names and descriptions of products. This means that if the backend receives a corresponding POST request with parameters (or JSON data in case of a REST API) *product_name* and *product_description* from a product manager, then the name and description of a specific product are changed. However, if the code that processes this request can also handle additional parameters, e.g., *product_price*, a malicious product manager can illegitimately modify a product (here its price) by learning or guessing the corresponding parameter name(s)[2].

This challenge shows you that it's important to be careful when processing data received in requests and to make sure that only data that should be processed (given the rights of the current user) are actually processed. Based on this understanding, it shouldn't be too difficult to prevent mass assignment vulnerabilities.

2.2 Required solution for full points

- A description of the attack you used to change the account of a standard user to an *admin* account.
- A screenshot of the *ADMIN PAGE* you get as user *alice* to demonstrate the attack was executed successfully.

2.3 Hints

- In this challenge, you are using the *Chocoshop* for the first time. Work with user *alice* (password *alice.123*), which is a standard user of the shop. After the successful attack, the account of *alice* should be an *admin* account.
- This challenge is a good opportunity to learn more about *Burp*. Use *Burp* as a proxy when accessing the Chocoshop. In the *Proxy* tab and by selecting *HTTP history*, you can inspect the requests/responses that have been sent/received so far. Also, by right-clicking a specific request and selecting *Send to Repeater*, this request is sent to the *Repeater* tab. There, you can modify the request, send it to the web application with the *Send* button, and directly inspect the resulting response.
- The vulnerability is located on the profile page (which you can access at the top by clicking *PROFILE* after having logged in). Specifically, the vulnerability is associated with the functionality to edit the personal information on the profile page.
- As a final hint, analyze the requests/responses that are sent/received when accessing the profile page and when updating the personal information.
- Once you have executed the attack, access the base page of the Chocoshop. This should show the admin page.

[2] Additional information about mass assignment attacks can be found here: *https://cheatsheetseries.owasp.org/cheatsheets/Mass_Assignment_Cheat_Sheet.html*

3 Password Spraying HTTP

3.1 Goal

Get the username (in the range *user_140000* - *user_140500*) that has a specific given password by carrying out an HTTP password spraying attack.

This challenge shows you that sometimes, attackers or penetration testers have to be creative to get around security defenses that are implemented by a system. In the case of this challenge, these defenses temporarily block the user after a few failed logins with a specific username or after a few failed logins from the same source IP address. Also, this challenge shows you that good attackers or penetration testers should be aware of available helpful tools to get around the defenses and that it is sometimes necessary to write some code (e.g., a shell script) to easily combine available tools to efficiently carry out an attack.

Before working on the challenge, watch the two videos linked under *Theory* in the challenge description. Start with one about PW Spraying (which is in German) and next, watch the one about changing the source IP address. In case you don't know Tor[3] (which is used in the second video): Tor (standing for *The Onion Router*) is an overlay network that allows anonymous Internet connections by relaying data over multiple hops. As a result of this, the service a user is contacting does not see the source IP address of the user, but only the address of the final hop. Note that Tor is relatively slow, so be patient when using it in this challenge. Also note that if you are trying out the commands used in the second video, use *ifconfig.io* instead of *ipconfig.io*, as *ifconfig.io* works better.

Note also that the *Resources Properties* are a bit misleading. The link *https://pwspray.vm.vuln.land* shows the info page with the currently valid passwords. The actual login is reachable at *http://pwspray.vm.vuln.land* (*http*, not *https*). You can ignore the *ssh/ftp* services, we only deal with http in this challenge.

3.2 Required solution for full points

- The code of the script / program you used to execute the attack (paste it or attach it as a file).
- The username/password pair that you managed to uncover.

3.3 Hints

- As you can see from the login popup window when accessing *http://pwspray.vm.vuln.land*, HTTP basic authentication is used. In this case, the credentials are included in the request in an *Authorization* header, in Base64 encoded form. Specifically, username and password are first combined with a ":", e.g., *user_140345:94ecbf29*, next this is Base64-encoded to, e.g., *dXNlcl8xNDAzNDU6OTRlY2JmMjk=*, and the resulting header in the request would be *Authorization: Basic dXNlcl8xNDAzNDU6OTRlY2JmMjk=*.
- If the username/password combination is valid, the first header of the response is *HTTP/1.1 200 OK*. Otherwise, it is *HTTP/1.1 401 Unauthorized.*
- From Moodle, you can download a skeleton for a shell script (*httpspray.sh*), which you can use as a basis for the attack. The script takes a password candidate as a command line parameter and then tries all 501 possible usernames with this password. Call it as follows: *./httpspray.sh 94ecbf29*, where *94ecbf29* is the password to use. Of course, you can also use any other technology to write such a program, but it's certainly a good idea to get familiar with shell scripting. Study this skeleton script so you understand in detail what it is doing. As you should recognize, it already contains code to iterate through all possible usernames and it «already gives you the opportunity» to (re)start Tor at the beginning and after 10 login attempts. Complete the script so that it works and use it to do the attack.

[3] *https://www.torproject.org*

- The following Linux commands may be helpful:
  - Restarting Tor can be done with *sudo systemctl restart tor* (this is the easier approach than the one presented in the video).
  - Assuming the desired *username:password* value is stored in *candidate*, the following command can be used to send the login request over Tor and to return the full HTTP response, including the response header (*-u* includes an Authorization header with the correct Base64-encoding):
    proxychains -q curl -s --include http://pwspray.vm.vuln.land -u "$candidate"
  - To store the output of a command (such as, e.g., the HTTP response in the case of the command above) in a variable *output*, use:
    output=$(command)

Lab Points

In this lab, you can get **2 Lab Points**. To get them, you must achieve all 400 Hacking-Lab points that you can get from solving the challenges in this lab.